Block-Diagonal and LT Codes for Distributed Computing With Straggling Servers

Albin Severinson, *Student Member, IEEE*, Alexandre Graell i Amat, *Senior Member, IEEE*, and Eirik Rosnes, *Senior Member, IEEE*

***Abstract*—We propose two coded schemes for the distributed computing problem of multiplying a matrix by a set of vectors. The first scheme is based on partitioning the matrix into submatrices and applying maximum distance separable (MDS) codes to each submatrix. For this scheme, we prove that up to a given number of partitions the communication load and the computational delay (not including the encoding and decoding delay) are identical to those of the scheme recently proposed by Li *et al.*, based on a single, long MDS code. However, due to the use of shorter MDS codes, our scheme yields a significantly lower overall computational delay when the delay incurred by encoding and decoding is also considered. We further propose a second coded scheme based on Luby Transform (LT) codes under inactivation decoding. Interestingly, LT codes may reduce the delay over the partitioned scheme at the expense of an increased communication load. We also consider distributed computing under a deadline and show numerically that the proposed schemes outperform other schemes in the literature, with the LT code-based scheme yielding the best performance for the scenarios considered.**

***Index Terms*—Block-diagonal coding, computational delay, decoding delay, distributed computing, Luby Transform codes, machine learning algorithms, maximum distance separable codes, straggling servers.**

I. Introduction

Distributed computing systems have emerged as one of the most effective ways of solving increasingly complex computational problems, such as those in large-scale machine learning and data analytics [1]–[3]. These systems, referred to as "warehouse-scale computers" (WSCs) [1], may be composed of thousands of relatively homogeneous hardware and software components. Achieving high availability and efficiency for applications running on WSCs is a major challenge. One of the main reasons is the large number of components that may experience transient or permanent failures [3]. As a result, several distributed computing frameworks have been proposed [4]–[6]. In particular, MapReduce [4] has gained significant attention as a means of effectively utilizing large computing clusters. For example, Google routinely performs computations over several thousands of servers using MapReduce [4]. Among the challenges brought on by distributed computing systems, the problems of straggling servers and bandwidth scarcity have recently received significant attention. The straggler problem is a synchronization problem characterized by the fact that a distributed computing task must wait for the slowest server to complete its computation, which may cause large delays [4]. On the other hand, distributed computing tasks typically require that data is moved between servers during the computation, the so-called *data shuffling*, which is a challenge in bandwidth-constrained networks.

Coding for distributed computing to reduce the computational delay and the communication load between servers has recently been considered in [7], [8]. In [7], a structure of repeated computation tasks across servers was proposed, enabling coded multicast opportunities that significantly reduce the required bandwidth to shuffle the results. In [8], the authors showed that maximum distance separable (MDS) codes can be applied to a linear computation task (e.g., multiplying a vector with a matrix) to alleviate the effects of straggling servers and reduce the computational delay. In [9], a unified coding framework was presented and a fundamental tradeoff between computational delay and communication load was identified. The ideas of [7], [8] can be seen as particular instances of the framework in [9], corresponding to the minimization of the communication load and the computational delay, respectively. The code proposed in [9] is an MDS code of code length proportional to the number of rows of the matrix to be multiplied, which may be very large in practice. For example, Google performs matrix-vector multiplications with matrices of dimension of the order of $10^{10} \times 10^{10}$ when ranking the importance of websites [10]. In [7]–[9], the computational delay incurred by the encoding and decoding is not considered. However, the encoding and decoding may incur a high computational delay for large matrices.

Coding has previously been applied to several related problems in distributed computing. For example, the scheme in [8] has been extended to distributed matrix-matrix multiplication where both matrices are too large to be stored at one server [11], [12]. Whereas the schemes in [8], [11] are based on MDS codes, the scheme in [12] is based on a novel coding scheme that exploits the algebraic properties of matrix-matrix multiplication over a finite field to reduce the computational

This work was presented in part at the IEEE Information Theory Workshop (ITW), Kaohsiung, Taiwan, November 2017, and it was published in part in "Coding for Distributed Computing: Investigating and improving upon coding theoretical frameworks for distributed computing," MSc. thesis, Chalmers University of Technology, June 2017. This work was funded by the Swedish Research Council under grant 2016-04253 and the Research Council of Norway under grant 240985/F20.

Albin Severinson was with the Department of Electrical Engineering, Chalmers University of Technology, SE-41296 Gothenburg, Sweden. He is now with Simula UiB and the Department of Informatics at the University of Bergen, N-5020 Bergen, Norway (email: albin@severinson.org).

Alexandre Graell i Amat is with the Department of Electrical Engineering, Chalmers University of Technology, SE-41296 Gothenburg, Sweden (email: alexandre.graell@chalmers.se).

Eirik Rosnes is with Simula UiB, N-5020 Bergen, Norway (email: eirikrosnes@simula.no).

delay. In [13], it was shown that introducing sparsity in a structured manner during encoding can speed up computing dot products between long vectors. Distributed computing over heterogeneous clusters has been considered in [14].

In this paper, we propose two coding schemes for the problem of multiplying a matrix by a set of vectors. The first is a block-diagonal coding (BDC) scheme equivalent to partitioning the matrix and applying smaller MDS codes to each submatrix separately (we originally introduced the BDC scheme in [15]). The storage design for the BDC scheme can be cast as an integer optimization problem, whose computation scales exponentially with the problem size. We propose a heuristic solver for efficiently solving the optimization problem, and a branch-and-bound approach for improving on the resulting solution iteratively. Furthermore, we prove that up to a certain level of partitioning the BDC scheme has identical computational delay (as defined in [9]) and communication load to those of the scheme in [9]. Interestingly, when the delay incurred by encoding and decoding is taken into account, the proposed scheme achieves an overall computational delay significantly lower than that of the scheme in [9]. We further propose a second coding scheme based on Luby Transform (LT) codes [16] under inactivation decoding [17], which in some scenarios achieves a lower computational delay than that of the BDC scheme at the expense of a higher communication load. We show that for the LT code-based scheme it is possible to trade an increase in communication load for a lower computational delay. We finally consider distributed computing under a deadline, where we are interested in completing a computation within some computational delay, and show numerically that both the BDC and the LT code-based schemes significantly increase the probability of meeting a deadline over the scheme in [9]. In particular, the LT code-based scheme achieves the highest probability of meeting a deadline for the scenarios considered.

II. System Model and Preliminaries

We consider the distributed matrix multiplication problem, i.e., the problem of multiplying a set of vectors with a matrix. In particular, given an $m \times n$ matrix $\boldsymbol{A} \in \mathbb{F}_{2^l}^{m\times n}$ and N vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in \mathbb{F}_{2^l}^{n}$, where $\mathbb{F}_{2^l}$ is an extension field of characteristic 2, we want to compute the N vectors $\boldsymbol{y}_1 = \boldsymbol{A}\boldsymbol{x}_1, \dots, \boldsymbol{y}_N = \boldsymbol{A}\boldsymbol{x}_N$. The computation is performed in a distributed fashion using K servers, $S_1, \dots, S_K$. Each server is responsible for multiplying ηm matrix rows by the vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$, for some $\frac{1}{K} \leq \eta \leq 1$. We refer to η as the fraction of rows stored at each server and we assume that η is selected such that ηm is an integer. Prior to computing $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$, $\boldsymbol{A}$ is encoded by an $r \times m$ encoding matrix $\boldsymbol{\Psi} = [\Psi_{i,j}]$, resulting in the coded matrix $\boldsymbol{C} = \boldsymbol{\Psi}\boldsymbol{A}$, of size $r \times n$, i.e., the rows of $\boldsymbol{A}$ are encoded using an (r, m) linear code with $r \geq m$. This encoding is carried out in a distributed manner over the K servers and is used to alleviate the straggler problem. We allow assigning each row of the coded matrix $\boldsymbol{C}$ to several servers to enable coded multicasting, a strategy used to address the bandwidth scarcity problem. Let

$$q = K\frac{m}{r},$$

where we assume that r divides Km and hence q is an integer. The r coded rows of $\boldsymbol{C}$, $\boldsymbol{c}_1, \dots, \boldsymbol{c}_r$, are divided into $\binom{K}{\eta q}$ disjoint batches, each containing $r/\binom{K}{\eta q}$ coded rows. Each batch is assigned to ηq servers. Correspondingly, a batch B is labeled by a unique set $\mathcal{T} \subset \{S_1, \dots, S_K\}$, of size $|\mathcal{T}| = \eta q$, denoting the subset of servers that store that batch. We write $B_{\mathcal{T}}$ to denote the batch stored at the unique set of servers $\mathcal{T}$. Server S_k, $k = 1, \dots, K$, stores the coded rows of $B_{\mathcal{T}}$ if and only if $S_k \in \mathcal{T}$.

A. Probabilistic Runtime Model

We assume that running a computation on a single server takes a random amount of time, which is denoted by the random variable H, according to the shifted-exponential cumulative probability distribution function (CDF)

$$F_H(h;\sigma) = \begin{cases} 1 - \mathrm{e}^{-\left(\frac{h}{\sigma}-1\right)}, & \text{for } h \geq \sigma \\ 0, & \text{otherwise} \end{cases},$$

where σ is a parameter used to scale the distribution. Denote by σ_{A} and σ_{M} the number of time units required to complete one addition and one multiplication (over $\mathbb{F}_{2^l}$), respectively, over a single server. Let σ be the weighted sum of the number of additions and multiplications required to complete the computation, where the weighting coefficients are σ_{A} and σ_{M}, respectively. As in [18], we assume that σ_{A} is in $\mathcal{O}(\frac{l}{64})$ and σ_{M} in $\mathcal{O}(l \log_2 l)$. Furthermore, we assume that the hidden coefficients are comparable and will thus not consider them. With some abuse of language, we refer to the parameter σ associated with some computation as its computational complexity. For example, the complexity (number of time units) of computing the inner product of two length-n vectors is $\sigma = (n-1)\sigma_{\mathsf{A}} + n\sigma_{\mathsf{M}}$ as it requires performing $n-1$ additions and n multiplications. The shift of the shifted-exponential distribution should be interpreted as the minimum amount of time the computation can be completed in. The tail of the distribution accounts for transient disturbances that are at the root of the straggler problem. These include transmission and queuing delays during initialization as well as contention for the local disk and slow-downs due to higher priority tasks being assigned to the same server [19]. The complexity of a computation σ affects both the shift and the tail of the distribution since the probability of transient behavior occurring increases with the amount of time the computation is running. In the results section we also consider a model where σ only affects the shift. The shifted-exponential distribution was proposed as a model for the latency of file queries from cloud storage systems in [20] and was subsequently used to model computational delay in [8], [9].

When an algorithm is split into K parallel subtasks that are run across K servers, we denote the runtime of the subtask running on server S_k by H_k. As in [8], we assume that $H_1, \dots, H_K$ are independent and identically distributed random variables with CDF $F_H(Kh;\sigma)$. For $i = 1, \dots, K$, we denote the i-th order statistic by $H_{(i)}$, i.e., the i-th smallest random variable of $H_1, \dots, H_K$. The runtime of the i-th fastest server to complete its subtask is thus given by

$H_{(i)}$, which is a Gamma distributed random variable with expectation and variance given by [21]

$$\mu(\sigma, K, i) \triangleq \mathbb{E}\left[H_{(i)}\right] = \sigma \left(1 + \sum_{j=K-i+1}^{K} \frac{1}{j}\right),$$

$$\mathrm{Var}\left[H_{(i)}\right] = \sigma^2 \sum_{j=K-i+1}^{K} \frac{1}{j^2}.$$

We parameterize the Gamma distribution by its inverse scale factor a and its shape parameter b. We give these in terms of the distribution mean and variance as [22]

$$a = \frac{\mathbb{E}[H_{(i)}] - \sigma}{\mathrm{Var}[H_{(i)}]} \text{ and } b = \frac{\left(\mathbb{E}[H_{(i)}] - \sigma\right)^2}{\mathrm{Var}[H_{(i)}]}.$$

Denote by $F_{H_{(i)}}(h_{(i)}; \sigma, K)$ the CDF of $H_{(i)}$. It is given by [22]

$$F_{H_{(i)}}(h_{(i)}; \sigma, K) = \begin{cases} \frac{\gamma(b, a(h_{(i)} - \sigma))}{\Gamma(b)}, & \text{for } h_{(i)} \geq \sigma \\ 0, & \text{otherwise} \end{cases},$$

where Γ denotes the Gamma function and γ the lower incomplete Gamma function,

$$\Gamma(b) = \int_0^\infty x^{b-1} \mathrm{e}^{-x} \, \mathrm{d}x \text{ and } \gamma(b, ah) = \int_0^{ah} x^{b-1} \mathrm{e}^{-x} \, \mathrm{d}x.$$

We remark that $F_{H_{(i)}}(h_{(i)}; \sigma, K)$ is the probability of a computation finishing prior to some deadline $t = h_{(i)}$.

B. Distributed Computing Model

We consider the coded computing framework introduced in [9], which extends the MapReduce framework [4]. The overall computation proceeds in three phases, the *map*, *shuffle*, and *reduce* phases, which are augmented to make use of the coded multicasting strategy proposed in [7] to address the bandwidth scarcity problem and the coded scheme proposed in [8] to alleviate the straggler problem. Furthermore, we consider the delay incurred by the encoding of $\boldsymbol{A}$ that takes place before the start of the map phase. We refer to this as the *encoding* phase. Also, we assume that the matrices $\boldsymbol{A}$ and $\boldsymbol{\Psi}$ as well as the input vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$ are known to all servers at the start of the computation. The overall computation proceeds in the following manner.

1) Encoding Phase: In the encoding phase, the coded matrix $\boldsymbol{C}$ is computed from $\boldsymbol{A}$ and $\boldsymbol{\Psi}$ in a distributed fashion. Specifically, denote by $\mathcal{R}^{(S)}$ the set of indices of rows of $\boldsymbol{C}$ that are assigned to server S and denote by $\boldsymbol{\Psi}^{(S)}$ the matrix consisting of the rows of $\boldsymbol{\Psi}$ with indices from $\mathcal{R}^{(S)}$. Then, server S computes the coded rows it needs by multiplying $\boldsymbol{\Psi}^{(S)}$ by $\boldsymbol{A}$. Note that since we assign each coded row to ηq servers, each row of $\boldsymbol{C}$ is computed separately by ηq servers. We define the computational delay of the encoding phase as its average runtime per source row and vector $\boldsymbol{y}$, i.e.,

$$D_{\mathsf{encode}} = \frac{\eta q}{mN} \mu\left(\frac{\sigma_{\mathsf{encode}}}{K}, K, K\right),$$

where σ_{encode} is the complexity of the encoding. During the encoding process, the rows of $\boldsymbol{\Psi}$ are multiplied by the columns of $\boldsymbol{A}$. Therefore, the complexity scales with the product of the number of nonzero elements of $\boldsymbol{\Psi}$ and the number of columns of $\boldsymbol{A}$. Specifically,

$$\sigma_{\mathsf{encode}} = |\{(i,j) : \Psi_{i,j} \neq 0\}| \, n \left(\sigma_{\mathsf{A}} + \sigma_{\mathsf{M}}\right) - n\sigma_{\mathsf{A}}.$$

Alternatively, we compute $\boldsymbol{C}$ by performing a decoding operation on $\boldsymbol{A}$. In this case σ_{encode} is the decoding complexity (see Section IV-B). Furthermore, since the decoding algorithms are designed to decode the entire codeword, each server has to compute all rows of $\boldsymbol{C}$. Using this strategy the encoding delay is

$$D_{\mathsf{encode}} = \frac{K}{mN} \mu\left(\frac{\sigma_{\mathsf{encode}}}{K}, K, K\right).$$

For each case we choose the strategy that minimizes the delay.

2) Map Phase: In the map phase, we compute in a distributed fashion coded intermediate values, which will be later used to obtain vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$. Server S multiplies the input vectors $\boldsymbol{x}_j$, $j = 1, \dots, N$, by all the coded rows of matrix $\boldsymbol{C}$ it stores, i.e., it computes

$$\mathcal{Z}_j^{(S)} = \{\boldsymbol{c}\boldsymbol{x}_j : \, \boldsymbol{c} \in \{B_{\mathcal{T}} : \, S \in \mathcal{T}\}\}, \; j = 1, \dots, N.$$

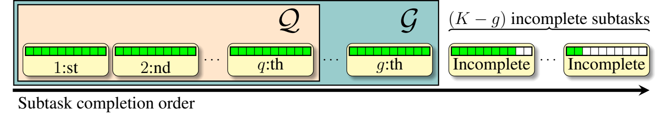

Fig. 1. Servers (yellow boxes) finish their respective subtasks in random order.

The map phase terminates when a set of servers $\mathcal{G} \subseteq \{S_1, \dots, S_K\}$ that collectively store enough values to decode the output vectors have finished their map computations. We denote the cardinality of $\mathcal{G}$ by g. The (r, m) linear code proposed in [9] is an MDS code for which $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ can be obtained from any subset of q servers, i.e., $g = q$. We illustrate the completion of subtasks in Fig. 1.

We define the computational delay of the map phase as its average runtime per source row and vector $\boldsymbol{y}$, i.e.,

$$D_{\mathsf{map}} = \frac{1}{mN} \mu\left(\frac{\sigma_{\mathsf{map}}}{K}, K, g\right),$$

where $\sigma_{\mathsf{map}} = K\eta mN \left((n-1)\sigma_{\mathsf{A}} + n\sigma_{\mathsf{M}}\right)$, as all K servers compute ηm inner products, each requiring $n-1$ additions and n multiplications, for each of the N input vectors. In [9], D_{map} is referred to simply as the computational delay.

After the map phase, the computation of $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ proceeds using only the servers in $\mathcal{G}$. We denote by $\mathcal{Q} \subseteq \mathcal{G}$ the set of the first q servers to complete the map phase. Each of the q servers in $\mathcal{Q}$ is responsible to compute N/q of the vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$. Let $\mathcal{W}_S$ be the set containing the indices of the vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ that server $S \in \mathcal{Q}$ is responsible for. The remaining servers in $\mathcal{G}$ assist the servers in $\mathcal{Q}$ in the shuffle phase.

3) Shuffle Phase: In the shuffle phase, intermediate values calculated in the map phase are exchanged between servers in $\mathcal{G}$ until all servers in $\mathcal{Q}$ hold enough values to compute the vectors they are responsible for. As in [9], we allow creating and multicasting coded messages that are simultaneously useful for multiple servers. Furthermore, as in [8], we denote by

$\phi(j)$ the ratio between the communication load of unicasting the same message to each of j recipients and multicasting that message to j recipients. For example, if the communication load of multicasting a message to j recipients and unicasting a message to a single recipient is the same, we have $\phi(j) = j$. On the other hand, if the communication load of multicasting a message to j recipients is equal to that of unicasting that same message to each recipient, $\phi(j) = 1$. The total communication load of a multicast message is then given by $\frac{j}{\phi(j)}$. The shuffle phase proceeds in three steps as follows.

1) Coded messages composed of several intermediate values are multicasted among the servers in $\mathcal{Q}$.
2) Intermediate values are unicasted among the servers in $\mathcal{Q}$.
3) Any intermediate values still missing from servers in $\mathcal{Q}$ are unicasted from the remaining servers in $\mathcal{G}$, i.e., from the servers in $\mathcal{G} \setminus \mathcal{Q}$.

For a subset of servers $\mathcal{S} \subset \mathcal{Q}$ and $S \in \mathcal{Q} \setminus \mathcal{S}$, we denote the set of intermediate values needed by server S and known *exclusively* by the servers in $\mathcal{S}$ by $\mathcal{V}_{\mathcal{S}}^{(S)}$. More formally,

$$\mathcal{V}_{\mathcal{S}}^{(S)} \triangleq \{\boldsymbol{c}\boldsymbol{x}_j : \ j \in \mathcal{W}_S \text{ and } \boldsymbol{c} \in \{B_{\mathcal{T}} : \ \mathcal{T} \cap \mathcal{Q} = \mathcal{S}\}\}.$$

We transmit coded multicasts only between the servers in $\mathcal{Q}$, and each coded message is simultaneously sent to multiple servers. We denote by

$$s_q \triangleq \inf\left(s : \sum_{j=s}^{\eta q} \alpha_j \leq 1 - \eta\right), \ \alpha_j \triangleq \frac{\binom{q-1}{j}\binom{K-q}{\eta q - j}}{\frac{q}{K}\binom{K}{\eta q}}, \quad (1)$$

the smallest number of recipients of a coded message [9]. We remark that $m\alpha_j$ is the total number of coded values delivered to each server via the coded multicast messages with exactly j recipients. More specifically, for each $j \in \{\eta q, \eta q - 1, \ldots, s_q\}$, and every subset $\mathcal{S} \subseteq \mathcal{Q}$ of size $j+1$, the shuffle phase proceeds as follows.

1) For each $S \in \mathcal{S}$, we evenly and arbitrarily split $\mathcal{V}_{\mathcal{S} \setminus S}^{(S)}$ into j disjoint segments, $\mathcal{V}_{\mathcal{S} \setminus S}^{(S)} = \{\mathcal{V}_{\mathcal{S} \setminus S, \tilde{S}}^{(S)} : \tilde{S} \in \mathcal{S} \setminus S\}$, and associate the segment $\mathcal{V}_{\mathcal{S} \setminus S, \tilde{S}}^{(S)}$ to server $\tilde{S}$.
2) Server $\tilde{S} \in \mathcal{S}$ multicasts the bit-wise modulo-2 sum of all the segments associated to it in $\mathcal{S}$. More precisely, it multicasts $\oplus_{S \in \mathcal{S} \setminus \tilde{S}} \mathcal{V}_{\mathcal{S} \setminus S, \tilde{S}}^{(S)}$ to the other servers in $\mathcal{S} \setminus \tilde{S}$, where $\oplus$ denotes the modulo-2 sum operator.

By construction, exactly one value that each coded message is composed of is unknown to each recipient. The other values have been computed locally by the recipient. More precisely, for every pair of servers $S, \tilde{S} \in \mathcal{S}$, since server S has computed locally the segments $\mathcal{V}_{\mathcal{S} \setminus S', \tilde{S}}^{(S')}$ for all $S' \in \mathcal{S} \setminus \{\tilde{S}, S\}$, it can cancel them from the message sent by server $\tilde{S}$, and recover the intended segment. We finish the shuffle phase by either unicasting any remaining needed values until all servers in $\mathcal{Q}$ hold enough intermediate values to decode successfully, or by repeating the above two steps for $j = s_q - 1$. We refer to these alternatives as shuffling strategy 1 and 2, respectively. We always select the strategy achieving the lowest communication load. If any server in $\mathcal{Q}$ still needs more intermediate values at this point, they are unicasted from other servers in $\mathcal{G}$. This may happen only if a non-MDS code is used. We remark that it may be possible to opportunistically create additional coded multicasting opportunities by exploiting the remaining $g - q$ servers in $\mathcal{G}$.

Definition 1. *The communication load, denoted by L, is the number of unicasts and multicasts (weighted by their cost relative to a unicast) per source row and vector $\boldsymbol{y}$ exchanged during the shuffle phase. Specifically, each unicasted message increases L by $\frac{1}{mN}$, and each message multicasted to j recipients increases L by $\frac{j}{mN\phi(j)}$.*

The communication load after completing the shuffle phase is given in [9]. If the shuffle phase finishes by unicasting the remaining needed values (strategy 1), the communication load after completing the multicast phase is

$$\sum_{j=s_q}^{\eta q} \frac{\alpha_j}{\phi(j)}.$$

If instead steps 1) and 2) are repeated for $j = s_q - 1$ (strategy 2), the communication load is

$$\sum_{j=s_q-1}^{\eta q} \frac{\alpha_j}{\phi(j)}.$$

For the scheme in [9], the total communication load is

$$L_{\mathsf{MDS}} = \min\left(\sum_{j=s_q}^{\eta q} \frac{\alpha_j}{\phi(j)} + 1 - \eta - \sum_{j=s_q}^{\eta q} \alpha_j, \sum_{j=s_q-1}^{\eta q} \frac{\alpha_j}{\phi(j)}\right), \quad (2)$$

where $1 - \eta - \sum_{j=s_q}^{\eta q} \alpha_j$ is the communication load due to unicasting the remaining needed values.

4) Reduce Phase: Finally, in the reduce phase, the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ are computed. More specifically, server $S \in \mathcal{Q}$ uses the locally computed sets $\mathcal{Z}_1^{(S)}, \ldots, \mathcal{Z}_N^{(S)}$ and the received messages to compute the vectors $\boldsymbol{y}_j$, $\forall j \in \mathcal{W}_S$. The computational delay of the reduce phase is its average runtime per source row and output vector $\boldsymbol{y}$, i.e.,

$$D_{\mathsf{reduce}} = \frac{1}{mN}\mu\left(\frac{\sigma_{\mathsf{reduce}}}{q}, q, q\right),$$

where σ_{reduce} is the computational complexity (see Section II-A) of the reduce phase.

Definition 2. *The overall computational delay, D, is the sum of the encoding, map, and reduce phase delays, i.e., $D = D_{\mathsf{encode}} + D_{\mathsf{map}} + D_{\mathsf{reduce}}$.*

C. Previously Proposed Coded Computing Schemes

Here we formally define the uncoded scheme (UC) and the coded computing schemes of [7]–[9] (which we refer to as the straggler coding (SC), coded MapReduce (CMR), and unified scheme, respectively) in terms of the model above. Specifically, to make a fair comparison with our coded computing scheme with parameters K, q, m, and η, we define the corresponding uncoded, CMR, SC, and unified schemes. When referring to the system parameters of a given scheme, we will write the scheme acronym in the subscript. We only

explicitly mention the parameters that differ. The number of servers K is unchanged for all schemes considered.

The uncoded scheme uses no erasure coding and no coded multicasting and has parameters $\eta_{\mathsf{UC}} = \frac{1}{K}$ and $q_{\mathsf{UC}} = K$, implying $\eta_{\mathsf{UC}} q_{\mathsf{UC}} = 1$. Furthermore, the encoding matrix $\boldsymbol{\Psi}_{\mathsf{UC}}$ is the $m \times m$ identity matrix and the coded matrix is $\boldsymbol{C}_{\mathsf{UC}} = \boldsymbol{A}$.

The CMR scheme [7] uses only coded multicasting, i.e., $\boldsymbol{C}_{\mathsf{CMR}} = \boldsymbol{A}$ and $q_{\mathsf{CMR}} = K$. Furthermore, the fraction of rows stored at each server is $\eta_{\mathsf{CMR}} = \frac{\eta q}{K}$. We remark that there is no reduce delay for this scheme, i.e., $D_{\mathsf{reduce}} = 0$.

The SC scheme [8] uses an erasure code but no coded multicasting. For the corresponding SC scheme, the code rate is unchanged, i.e., $q_{\mathsf{SC}} = q$, and the fraction of rows stored at each server is $\eta_{\mathsf{SC}} = \frac{1}{q_{\mathsf{SC}}}$. The encoding matrix $\boldsymbol{\Psi}_{\mathsf{SC}}$ of the SC scheme is obtained by splitting the rows of $\boldsymbol{A}$ into q_{SC} equally tall submatrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{q_{\mathsf{SC}}}$ and applying a (K, q_{SC}) MDS code to the elements of each submatrix, thereby creating K coded submatrices $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_K$. The coded matrix $\boldsymbol{C}_{\mathsf{SC}}$ is the concatenation of $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_K$, i.e.,

$$\boldsymbol{C}_{\mathsf{SC}} = \begin{pmatrix} \boldsymbol{C}_1 \\ \vdots \\ \boldsymbol{C}_K \end{pmatrix}.$$

The unified scheme [9] uses both an erasure code and coded multicasting and has parameters $\eta_{\mathsf{unified}} = \eta$ and $q_{\mathsf{unified}} = q$. Furthermore, the encoding matrix of the unified scheme, $\boldsymbol{\Psi}_{\mathsf{unified}}$, is an (r, m) MDS code encoding matrix.

III. Block-Diagonal Coding

In this section, we introduce a BDC scheme for the problem of multiplying a matrix by a set of vectors. For large matrices, the encoding and decoding complexity of the proposed scheme is significantly lower than that of the scheme in [9], leading to a lower overall computational delay, as will be shown in Section VII. Specifically, the scheme is based on a block-diagonal encoding matrix of the form

$$\boldsymbol{\Psi}_{\mathsf{BDC}} = \begin{bmatrix} \boldsymbol{\psi}_1 & & \\ & \ddots & \\ & & \boldsymbol{\psi}_T \end{bmatrix},$$

where $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_T$ are $\frac{r}{T} \times \frac{m}{T}$ encoding matrices of an $(\frac{r}{T}, \frac{m}{T})$ MDS code, for some integer T that divides m and r. Note that the encoding given by $\boldsymbol{\Psi}_{\mathsf{BDC}}$ amounts to partitioning the rows of $\boldsymbol{A}$ into T disjoint submatrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_T$ and encoding each submatrix separately. We refer to an encoding $\boldsymbol{\Psi}_{\mathsf{BDC}}$ with T disjoint submatrices as a T-partitioned scheme, and to the submatrix of $\boldsymbol{C} = \boldsymbol{\Psi}_{\mathsf{BDC}} \boldsymbol{A}$ corresponding to $\boldsymbol{\psi}_i$ as the i-th partition. We remark that all submatrices can be encoded using the same encoding matrix, i.e., $\boldsymbol{\psi}_i = \boldsymbol{\psi}$, $i = 1, \ldots, T$, reducing the storage requirements, and encoding/decoding can be performed in parallel if many servers are available. Notably, by keeping the ratio $\frac{m}{T}$ constant, the decoding complexity scales linearly with m. We further remark that the case $\boldsymbol{\Psi}_{\mathsf{BDC}} = \boldsymbol{\psi}$ (i.e., the number of partitions is $T = 1$) corresponds to the scheme in [9], which we will sometimes refer to as the *unpartitioned* scheme. We illustrate the BDC scheme with $T = 3$ partitions in Fig. 2.

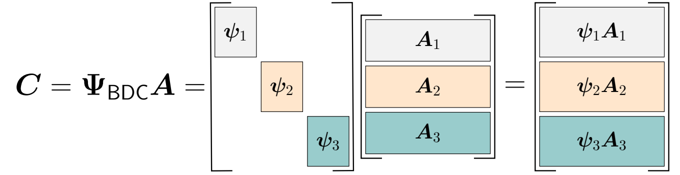

Fig. 2. BDC scheme with $T = 3$ partitions.

A. Assignment of Coded Rows to Batches

For a block-diagonal encoding matrix $\boldsymbol{\Psi}_{\mathsf{BDC}}$, we denote by $\boldsymbol{c}_i^{(t)}$, $t = 1, \ldots, T$ and $i = 1, \ldots, r/T$, the i-th coded row of $\boldsymbol{C}$ within partition t. For example, $\boldsymbol{c}_1^{(2)}$ denotes the first coded row of the second partition. As described in Section II, the coded rows are divided into $\binom{K}{\eta q}$ disjoint batches. To formally describe the assignment of coded rows to batches we use a $\binom{K}{\eta q} \times T$ integer matrix $\boldsymbol{P} = [p_{i,j}]$, where $p_{i,j}$ is the number of rows from partition j that are stored in batch i. In the sequel, $\boldsymbol{P}$ will be referred to as the assignment matrix. Note that, due to the MDS property, any set of m/T rows of a partition is sufficient to decode the partition. Thus, without loss of generality, we consider a *sequential* assignment of rows of a partition into batches. More precisely, when first assigning a row of partition t to a batch, we pick $\boldsymbol{c}_1^{(t)}$. Next time a row of partition t is assigned to a batch we pick $\boldsymbol{c}_2^{(t)}$, and so on. In this manner, each coded row is assigned to a unique batch exactly once. The rows of $\boldsymbol{P}$ are labeled by the subset of servers the corresponding batch is stored at, and the columns are labeled by their partition indices. For convenience, we refer to the pair $(\boldsymbol{\Psi}_{\mathsf{BDC}}, \boldsymbol{P})$ as the *storage design*. The assignment matrix $\boldsymbol{P}$ must satisfy the following conditions.

1) The entries of each row of $\boldsymbol{P}$ must sum up to the batch size, i.e.,

$$\sum_{j=1}^{T} p_{i,j} = \frac{r}{\binom{K}{\eta q}}, \ 1 \leq i \leq \binom{K}{\eta q}.$$

2) The entries of each column of $\boldsymbol{P}$ must sum up to the number of rows per partition, i.e.,

$$\sum_{i=1}^{\binom{K}{\eta q}} p_{i,j} = \frac{r}{T}, \ 1 \leq j \leq T.$$

We clarify the assignment of coded rows to batches and the coded computing scheme in the following example.

Example 1 ($m = 20$, $N = 4$, $K = 6$, $q = 4$, $\eta = 1/2$, $T = 5$)**.** *For these parameters, there are $r/T = 6$ coded rows per partition, of which $m/T = 4$ are sufficient for decoding, and $\binom{K}{\eta q} = 15$ batches, each containing $r/\binom{K}{\eta q} = 2$ coded rows. We construct the storage design shown in Fig. 3 with*

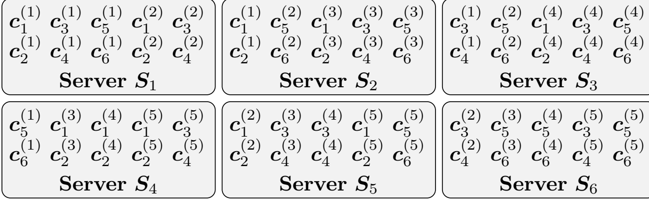

Fig. 3. Storage design for $m = 20$, $N = 4$, $K = 6$, $q = 4$, $\eta = 1/2$, and $T = 5$.

$\binom{K}{\eta q} \times T = 15 \times 5$ *assignment matrix*

$$\boldsymbol{P} = \begin{array}{c} \\ (S_1,S_2) \\ (S_1,S_3) \\ (S_1,S_4) \\ (S_1,S_5) \\ \vdots \\ (S_4,S_6) \\ (S_5,S_6) \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} \end{array}, \tag{3}$$

where rows are labeled by the subset of servers the batch is stored at, and columns are labeled by the partition index. In this case rows $\boldsymbol{c}_1^{(1)}$ *and* $\boldsymbol{c}_2^{(1)}$ *are assigned to batch* 1*,* $\boldsymbol{c}_3^{(1)}$ *and* $\boldsymbol{c}_4^{(1)}$ *are assigned to batch* 2*, and so on. For this storage design, any* $g = 4$ *servers collectively store at least* 4 *coded rows from each partition. However, some servers store more rows than needed to decode some partitions, suggesting that this storage design is suboptimal.*

Assume that $\mathcal{G} = \{S_1, S_2, S_3, S_4\}$ *is the set of* $g = 4$ *servers that finish their map computations first. Also, assign vector* $\boldsymbol{y}_i$ *to server* S_i*,* $i = 1, 2, 3, 4$*. We illustrate the coded shuffling scheme for* $\mathcal{S} = \{S_1, S_2, S_3\}$ *in Fig. 4. Server* S_1 *multicasts* $\boldsymbol{c}_1^{(1)}\boldsymbol{x}_3 \oplus \boldsymbol{c}_3^{(1)}\boldsymbol{x}_2$ *to* S_2 *and* S_3*. Since* S_2 *and* S_3 *can cancel* $\boldsymbol{c}_1^{(1)}\boldsymbol{x}_3$ *and* $\boldsymbol{c}_3^{(1)}\boldsymbol{x}_2$*, respectively, both servers receive one needed intermediate value. Similarly,* S_2 *multicasts* $\boldsymbol{c}_2^{(1)}\boldsymbol{x}_3 \oplus \boldsymbol{c}_5^{(2)}\boldsymbol{x}_1$*, while* S_3 *multicasts* $\boldsymbol{c}_4^{(1)}\boldsymbol{x}_2 \oplus \boldsymbol{c}_6^{(2)}\boldsymbol{x}_1$*. This process is repeated for* $\mathcal{S} = \{S_2, S_3, S_4\}$*,* $\mathcal{S} = \{S_1, S_3, S_4\}$*, and* $\mathcal{S} = \{S_1, S_2, S_4\}$*. After the shuffle phase, we have sent* 12 *multicast messages and* 30 *unicast messages, resulting in a communication load of* $(12 + 30)/20/4 = 0.525$*, a* 50% *increase from the load of the unpartitioned scheme (*0.35*, given by* (2)*). In this case,* S_1 *received additional intermediate values from partition* 2*, despite already storing enough, further indicating that the assignment in* (3) *is suboptimal.*

IV. Performance of the Block-Diagonal Coding

In this section, we analyze the impact of partitioning on the performance. We also prove that we can partition up to the batch size, i.e., $T = r/\binom{K}{\eta q}$, without increasing the communication load and the computational delay of the map phase with respect to the original scheme in [9].

A. Communication Load

For the unpartitioned scheme of [9], $\mathcal{G} = \mathcal{Q}$, and the number of remaining values that need to be unicasted after the multicast phase is constant regardless which subset $\mathcal{Q}$

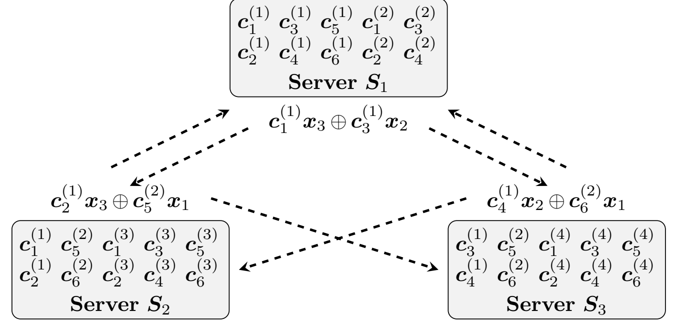

Fig. 4. Multicasting coded values between servers S_1, S_2, and S_3.

of servers finish first their map computations. However, for the BDC (partitioned) scheme, both g and the number of remaining unicasts may vary.

For a given assignment matrix $\boldsymbol{P}$ and a specific $\mathcal{Q}$, we denote by $U_{\mathcal{Q}}^{(S)}(\boldsymbol{P})$ the number of remaining values needed after the multicast phase by server $S \in \mathcal{Q}$, and by

$$U_{\mathcal{Q}}(\boldsymbol{P}) \triangleq \sum_{S \in \mathcal{Q}} U_{\mathcal{Q}}^{(S)}(\boldsymbol{P}) \tag{4}$$

the total number of remaining values needed by the servers in $\mathcal{Q}$. Note that both $U_{\mathcal{Q}}^{(S)}(\boldsymbol{P})$ and $U_{\mathcal{Q}}(\boldsymbol{P})$ depend on the strategy used to finish the shuffle phase (see Section II-B3). We remark that all sets $\mathcal{Q}$ are equally likely. Let $\mathbb{Q}^q$ denote the superset of all sets $\mathcal{Q}$. Furthermore, we denote by $L_{\mathbb{Q}}(\boldsymbol{P})$ the average communication load of the messages that are unicasted after the multicasting step (see Section II-B3), i.e.,

$$L_{\mathbb{Q}}(\boldsymbol{P}) \triangleq \frac{1}{mN} \frac{1}{|\mathbb{Q}^q|} \sum_{\mathcal{Q} \in \mathbb{Q}^q} U_{\mathcal{Q}}(\boldsymbol{P}). \tag{5}$$

When needed we write $L_{\mathbb{Q}}^{(1)}(\boldsymbol{P})$ and $L_{\mathbb{Q}}^{(2)}(\boldsymbol{P})$, where the superscript denotes the strategy used to finish the shuffle phase. For a given storage design $(\boldsymbol{\Psi}_{\mathsf{BDC}}, \boldsymbol{P})$, the communication load of the BDC scheme is given by

$$L_{\mathsf{BDC}}(\boldsymbol{\Psi}_{\mathsf{BDC}}, \boldsymbol{P}) = \min\left(\sum_{j=s_q}^{\eta q} \frac{\alpha_j}{\phi(j)} + L_{\mathbb{Q}}^{(1)}(\boldsymbol{P}), \sum_{j=s_q-1}^{\eta q} \frac{\alpha_j}{\phi(j)} + L_{\mathbb{Q}}^{(2)}(\boldsymbol{P}) \right). \tag{6}$$

Note that the load due to the multicast phase is independent of the level of partitioning. Furthermore, for the unpartitioned scheme $L_{\mathbb{Q}}^{(2)} = 0$ by design.

We first explain how $U_{\mathcal{Q}}^{(S)}$ is evaluated. Let $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ be a vector of length T, where the t-th element is the number of intermediate values from partition t stored by server S at the end of the multicast phase. Furthermore, each row of $\boldsymbol{P}$ corresponds to a batch, and coded multicasting is made possible by storing each batch at multiple servers. The intermediate values transmitted during the multicast phase thus correspond to rows of $\boldsymbol{P}$. The vector $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ is then computed by adding some set of rows of $\boldsymbol{P}$. The indices of the rows to add depend on $\mathcal{Q}$ and S (see Section II-B3).

We denote by $\left(\boldsymbol{u}_{\mathcal{Q}}^{(S)}\right)_t$ the t-th element of the vector $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$. The number of values $U_{\mathcal{Q}}^{(S)}$ is given by adding the number of intermediate values still needed for each partition, i.e.,

$$U_{\mathcal{Q}}^{(S)} = \sum_{t=1}^{T} \max\left(\frac{m}{T} - \left(\boldsymbol{u}_{\mathcal{Q}}^{(S)}\right)_t, 0\right). \tag{7}$$

Its sum over all $S \in \mathcal{Q}$ gives $U_{\mathcal{Q}}(\boldsymbol{P})$ (see (4)). Averaging $U_{\mathcal{Q}}(\boldsymbol{P})$ over all $\mathcal{Q}$ and normalizing yields $L_{\mathbb{Q}}(\boldsymbol{P})$ (see (5)).

Example 2 (Computing $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$)**.** *We consider the same system as in Example 1. We again assume that $\mathcal{G} = \mathcal{Q} = \{S_1, S_2, S_3, S_4\}$ is the set of $g = q = 4$ servers that finish their map computations first. During the multicast phase server S_1 receives the intermediate values in $\mathcal{V}_{\mathcal{S}\setminus S_1}^{(S_1)}$ for all sets $\mathcal{S}$ of cardinality $j+1=3$ (see Section II-B3). In this case, we perform coded multicasting within the sets*

- $\mathcal{S} = \{S_1, S_2, S_3\}$, $\mathcal{V}_{\mathcal{S}\setminus S_1}^{(S_1)} = \{\boldsymbol{c}_5^{(2)}\boldsymbol{x}_1, \boldsymbol{c}_6^{(2)}\boldsymbol{x}_1\}$,
- $\mathcal{S} = \{S_1, S_2, S_4\}$, $\mathcal{V}_{\mathcal{S}\setminus S_1}^{(S_1)} = \{\boldsymbol{c}_1^{(3)}\boldsymbol{x}_1, \boldsymbol{c}_2^{(3)}\boldsymbol{x}_1\}$,
- $\mathcal{S} = \{S_1, S_3, S_4\}$, $\mathcal{V}_{\mathcal{S}\setminus S_1}^{(S_1)} = \{\boldsymbol{c}_1^{(4)}\boldsymbol{x}_1, \boldsymbol{c}_2^{(4)}\boldsymbol{x}_1\}$.

Note that $\mathcal{V}_{\{S_2,S_3\}}^{(S_1)}$ contains the intermediate values computed from the coded rows stored in the batch that labels the 6-th row of the assignment matrix $\boldsymbol{P}$. In the same manner, $\mathcal{V}_{\{S_2,S_4\}}^{(S_1)}$ and $\mathcal{V}_{\{S_3,S_4\}}^{(S_1)}$ correspond to rows 7 and 10 of $\boldsymbol{P}$, respectively. Furthermore, prior to the shuffle phase server S_1 stores the batches corresponding to rows 1 to 5 of $\boldsymbol{P}$. Thus, $\boldsymbol{u}_{\{S_1,S_2,S_3,S_4\}}^{(S_1)}$ is equal to the sum of rows $1,2,3,4,5,6,7,$ and 10 of $\boldsymbol{P}$. In this case, $\boldsymbol{u}_{\{S_1,S_2,S_3,S_4\}}^{(S_1)} = (6,6,2,2,0)$, and S_1 needs 8 more intermediate values, i.e., $U_{\{S_1,S_2,S_3,S_4\}}^{(S_1)} = 8$. Computing $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ for arbitrary $\mathcal{Q}$ and S then corresponds to summing the rows of $\boldsymbol{P}$ corresponding to batches either stored by server S prior to the shuffle phase or received by S in the multicast phase. The row indices are computed as explained in Section II-B3.

For a given $\boldsymbol{\Psi}_{\mathsf{BDC}}$, the assignment of rows into batches can be formulated as an optimization problem, where one would like to minimize $L_{\mathsf{BDC}}(\boldsymbol{\Psi}_{\mathsf{BDC}}, \boldsymbol{P})$ over all assignments $\boldsymbol{P}$. More precisely, the optimization problem is

$$\min_{\boldsymbol{P}\in\mathbb{P}} L_{\mathsf{BDC}}(\boldsymbol{\Psi}_{\mathsf{BDC}}, \boldsymbol{P}),$$

where $\mathbb{P}$ is the set of all assignments $\boldsymbol{P}$. This is a computationally complex problem, since both the complexity of evaluating the performance of a given assignment and the number of assignments scale exponentially in the problem size (there are $q\binom{K}{q}$ vectors $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$). We address the optimization of the assignment matrix $\boldsymbol{P}$ in Section V.

B. Computational Delay

We consider the delay incurred by the encoding, map, and reduce phases (see Definition 2). As in [9], we do not consider the delay incurred by the shuffle phase as the computations it requires are simple in comparison. Note that in [9] only D_{map} is considered, i.e., $D = D_{\mathsf{map}}$. However, one should not neglect the computational delay incurred by the encoding and reduce phases. Thus, we consider the overall computational delay

$$D = D_{\mathsf{encode}} + D_{\mathsf{map}} + D_{\mathsf{reduce}}.$$

The encoding delay D_{encode} is a function of the number of nonzero elements of $\boldsymbol{\Psi}_{\mathsf{BDC}}$. As there are at most $\frac{m}{T}$ nonzero elements in each row of a block-diagonal encoding matrix, for an encoding scheme with T partitions we have

$$\sigma_{\mathsf{encode,BDC}} \le \frac{m}{T} rn\sigma_{\mathsf{M}} + \left(\frac{m}{T} - 1\right) rn\sigma_{\mathsf{A}}. \tag{8}$$

The reduce phase consists of decoding the N output vectors and hence the delay it incurs depends on the underlying code and decoding algorithm. We assume that each partition is encoded using a Reed-Solomon (RS) code and is decoded using either the Berlekamp-Massey (BM) algorithm or the FFT-based algorithm proposed in [23], whichever yields the lowest complexity. To the best of our knowledge the algorithm proposed in [23] is the lowest complexity algorithm for decoding long RS codes. We measure the decoding complexity by its associated shifted-exponential parameter σ (see Section II-A).

The number of field additions and multiplications required to decode an $(r/T, m/T)$ RS code using the BM algorithm is $(r/T)\left(\xi(r/T) - 1\right)$ and $(r/T)^2\xi$, respectively, where ξ is the fraction of erased symbols [24]. With ξ upper bounded by $1 - \frac{q}{K}$ (the map phase terminates when a fraction of at least $\frac{q}{K}$ symbols from each partition is available), the complexity of decoding the T partitions for all N output vectors is upper bounded as

$$\sigma_{\mathsf{reduce,BDC}}^{\mathsf{BM}} \le N\left(\sigma_{\mathsf{A}}\left(\frac{r^2(1-\frac{q}{K})}{T} - r\right) + \sigma_{\mathsf{M}}\frac{r^2(1-\frac{q}{K})}{T}\right). \tag{9}$$

On the other hand, the FFT-based algorithm has complexity $\mathcal{O}(r\log r)$ [23]. We estimate the number of additions and multiplications required for a given code length r by fitting a curve of the form $a + br\log_2(cr)$, where (a,b,c) are coefficients, to empiric results derived from the authors' implementation of the algorithm. For additions the resulting parameters are $(2, 8.5, 0.867)$ and for multiplications they are $(2,1,4)$. The resulting curves diverge negligibly at the measured points. The total decoding complexity for the FFT-based algorithm is

$$\begin{aligned}\sigma_{\mathsf{reduce,BDC}}^{\mathsf{FFT}} = {} & NT\sigma_{\mathsf{A}}\left(2 + \frac{8.5r}{T}\log_2(0.867r/T)\right)\\ & + NT\sigma_{\mathsf{M}}\left(2 + \frac{r}{T}\log_2(4r/T)\right).\end{aligned} \tag{10}$$

The encoding and decoding complexity of the unified scheme in [9] is given by evaluating (8) and either (9) or (10) (whichever gives the lowest complexity), respectively, for $T = 1$. For the BDC scheme, by choosing T close to r we can thus significantly lower the delay of the encoding and reduce phases. On the other hand, the scheme in [8] uses codes of length proportional to the number of servers K. The encoding and decoding complexity of the SC scheme in [8] is thus given by evaluating (8) and either (9) or (10) for $T = \frac{m}{q}$.

C. Lossless Partitioning

Theorem 1. *For $T \le r/\binom{K}{\eta q}$, there exists an assignment matrix $\boldsymbol{P}$ such that the communication load and the com-*

putational delay of the map phase are equal to those of the unpartitioned scheme.

Proof: The computational delay of the map phase is equal to that of the unpartitioned scheme if any q servers hold enough coded rows to decode all partitions. For $T = r/\binom{K}{\eta q}$ we let $\boldsymbol{P}$ be a $\binom{K}{\eta q} \times T$ all-ones matrix and show that it has this property by repeating the argument from [9, Sec. IV.B] for each partition. In this case, any set of q servers collectively store $\frac{\eta q m}{T}$ rows from each partition, and since each coded row is stored by at most ηq servers, any q servers collectively store at least $\frac{\eta q m}{\eta q T} = \frac{m}{T}$ unique coded rows from each partition. The computational delay of the map phase is thus unchanged from the unpartitioned scheme. The communication load is unchanged if $U_{\mathcal{Q}}^{(S)}$ is equal to that of the unpartitioned scheme for all $\mathcal{Q}$ and S. The number of values needed $U_{\mathcal{Q}}^{(S)}$ is computed from $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ (see (7)), which is the sum of l rows of $\boldsymbol{P}$, for some integer l. For the all-ones assignment matrix, because all rows of $\boldsymbol{P}$ are identical, we have

$$U_{\mathcal{Q}}^{(S)} = T \max\left(\frac{m}{T} - l, 0\right) = \max\left(m - Tl, 0\right),$$

which is the number of remaining values for the unpartitioned scheme.

Next, we consider the case where $T < r/\binom{K}{\eta q}$. First, consider the case $T = r/\binom{K}{\eta q} - j$, for some integer j, $0 \leq j < \frac{r}{2\binom{K}{\eta q}}$. We first set all entries of $\boldsymbol{P}$ equal to 1. At this point, the total number of unique rows of $\boldsymbol{C}$ per partition stored by any set of q servers is at least

$$\frac{m}{r/\binom{K}{\eta q}} = \frac{m}{r/\binom{K}{\eta q} - j} \frac{r/\binom{K}{\eta q} - j}{r/\binom{K}{\eta q}} = \frac{m}{T} \frac{r/\binom{K}{\eta q} - j}{r/\binom{K}{\eta q}}. \quad (11)$$

The number of coded rows per partition that are not yet assigned is given by r/T multiplied by the fraction of partitions removed $\frac{j}{r/\binom{K}{\eta q}}$, i.e.,

$$\frac{1}{T} \frac{rj}{r/\binom{K}{\eta q}} = \frac{1}{T} \frac{m\frac{K}{q}j}{r/\binom{K}{\eta q}}. \quad (12)$$

We assign these rows to batches such that an equal number of coded rows is assigned to each of the K servers, which is always possible due to the limitations imposed by the system model. Any set of q servers will thus store a fraction q/K of these rows. The total number of unique coded rows per partition stored among any set of q servers is then lower bounded by the sum of (12) weighted by q/K and (11), i.e.,

$$\frac{m}{T}\left(\frac{r/\binom{K}{\eta q} - j}{r/\binom{K}{\eta q}} + \frac{\frac{K}{q}j}{r/\binom{K}{\eta q}} \frac{q}{K}\right) = \frac{m}{T},$$

showing that it is possible to decode all partitions using the coded rows stored over any set of q servers.

The communication load is unchanged with respect to the case where the number of partitions is $r/\binom{K}{\eta q}$ if and only if no server receives rows it does not need in the multicast phase. Due to decreasing the number of partitions from $r/\binom{K}{\eta q}$ to $T = r/\binom{K}{\eta q} - j$, we increase the number of coded rows needed to decode each partition by

$$\frac{m}{T} - \frac{m}{r/\binom{K}{\eta q}} = \frac{1}{T} \frac{mj}{r/\binom{K}{\eta q}}. \quad (13)$$

Furthermore, reducing the number of partitions increases the number of coded rows per partition stored among any set of q servers (see (12) and the following text) by

$$\frac{1}{T} \frac{mj}{r/\binom{K}{\eta q}}. \quad (14)$$

Note that the number of additional rows needed to decode each partition (see (13)) is greater than or equal to the number of additional rows stored among the q servers (see (14)). It is thus impossible that too many coded rows are delivered for any partition.

Second, we consider the case $T = \frac{r/\binom{K}{\eta q} - j}{i}$, where j is chosen as for the first case above and where i is a positive integer. Now, we first set all elements of $\boldsymbol{P}$ to i. At this point the number of unique rows of $\boldsymbol{C}$ per partition stored by any set of q servers is given by (11) multiplied by a factor i (since we set each element of $\boldsymbol{P}$ to i instead of one). Furthermore, the number of coded rows per partition that are not yet assigned is given by (12). Therefore, by using the same strategy as for $i = 1$ and assigning the remaining rows to batches such that an equal number of rows is assigned to each of the K servers, we are guaranteed that the communication load and the computational delay are unchanged also in this case. ■

V. Assignment Solvers

For $T \leq r/\binom{K}{\eta q}$ partitions, we can choose the assignment matrix $\boldsymbol{P}$ as described in the proof of Theorem 1. For the case where $T > r/\binom{K}{\eta q}$, we propose two solvers for the problem of assigning rows into batches: a heuristic solver that is fast even for large problem instances, and a hybrid solver combining the heuristic solver with a branch-and-bound solver. The branch-and-bound solver produces an optimal assignment but is significantly slower, hence it can be used as stand-alone only for small problem instances. We use a dynamic programming approach to speed up the branch-and-bound solver by caching $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ for all S and $\mathcal{Q} \in \mathbb{Q}^q$. We index each cached $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ by the batches it is computed from. Whenever $U_{\mathcal{Q}}^{(S)}$ drops to 0 due to assigning a row to a batch, we remove the corresponding $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ from the index. We also store a vector of length T with the i-th entry giving the number of vectors $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ that miss intermediate values from the i-th partition. Specifically, the i-th element of this vector is the number of vectors $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ for which the i-th element is less than $\frac{m}{T}$. This allows us to efficiently assess the impact on $L_{\mathbb{Q}}(\boldsymbol{P})$ due to assigning a row to some batch. Since $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ is of length T and because the cardinality of $\mathcal{Q}$ and $\mathbb{Q}^q$ is q and $\binom{K}{q}$, respectively, the memory required to keep this index scales as $\mathcal{O}\left(Tq\binom{K}{q}\right)$ and is thus only an option for small problem instances.

For all solvers, we first label the batches lexiographically and then optimize L_{BDC} in (6). For example, for $\eta q = 2$, we label the first batch by S_1, S_2, the second by S_1, S_3, and so

Algorithm 1: Heuristic Assignment

```
Input : P, d, K, T, and ηq
for 0 ≤ a < d·C(K, ηq) do
    i ← ⌊a/d⌋ + 1
    j ← (a mod T) + 1
    p_{i,j} ← p_{i,j} + 1
end
return P
```

on. The solvers are available under the Apache 2.0 license [25]. We remark that choosing $\boldsymbol{P}$ is similar to the problem of designing the coded matrices stored by each server in [12].

A. Heuristic Solver

The heuristic solver is inspired by the assignment matrices created by the branch-and-bound solver for small instances. It creates an assignment matrix $\boldsymbol{P}$ in two steps. We first set each entry of $\boldsymbol{P}$ to

$$Y \triangleq \left\lfloor \frac{r}{\binom{K}{\eta q} \cdot T} \right\rfloor,$$

thus assigning the first $\binom{K}{\eta q}Y$ rows of each partition to batches such that each batch is assigned YT rows. Let $d = r/\binom{K}{\eta q} - YT$ be the number of rows that still need to be assigned to each batch. The $r/T - \binom{K}{\eta q}Y$ rows per partition not assigned yet are assigned in the second step as shown in Algorithm 1.

Interestingly, for $T \leq r/\binom{K}{\eta q}$ the heuristic solver creates an assignment matrix satisfying the requirements outlined in the proof of Theorem 1. In the special case of $T = r/\binom{K}{\eta q}$, the all-ones matrix is produced.

B. Branch-and-Bound Solver

The branch-and-bound solver finds an optimal solution by recursively branching at each batch for which there is more than one possible assignment and considering all options. The solver is initially given an empty assignment matrix, i.e., an all-zeros $\binom{K}{\eta q} \times T$ matrix. For each branch, we lower bound the value of the objective function of any assignment in that branch and only investigate branches with possibly better assignments. The branch-and-bound operations given below are repeated until there are no more potentially better solutions to consider.

1) Branch: For the first row of $\boldsymbol{P}$ with remaining assignments, branch on every available assignment for that row. More precisely, find the smallest index i of a row of the assignment matrix $\boldsymbol{P}$ whose entries do not sum up to the batch size, i.e.,

$$\sum_{j=1}^{T} p_{i,j} < \frac{r}{\binom{K}{\eta q}}.$$

For row i, branch on incrementing the element $p_{i,j}$ by 1 for all columns (with index j) such that their entries do not sum up to the number of coded rows per partition, i.e.,

$$\sum_{i=1}^{\binom{K}{\eta q}} p_{i,j} < \frac{r}{T}.$$

2) Bound: We use a dynamic programming approach to lower bound L_{BDC} for a subtree. Specifically, for each row i and column j of $\boldsymbol{P}$, we store the number of vectors $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ that are indexed by row i and where the j-th element satisfies

$$\left(\frac{m}{T} - \left(\boldsymbol{u}_{\mathcal{Q}}^{(S)}\right)_j\right) > 0.$$

Assigning a coded row to a batch can at most reduce L_{BDC} by $1/\left(mN\left|\mathbb{Q}^q\right|\right)$ for each $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ indexed by that batch. We compute the bound by assuming that no $\boldsymbol{u}_{\mathcal{Q}}^{(S)}$ will be removed from the index for any subsequent assignment.

C. Hybrid Solver

The branch-and-bound solver can only be used by itself for small instances. However, it can be used to complete a *partial* assignment matrix, i.e., a matrix $\boldsymbol{P}$ for which not all rows have entries that sum up to the batch size. The branch-and-bound solver then completes the assignment optimally. We first find a candidate solution using the heuristic solver and then iteratively improve it using the branch-and-bound solver. In particular, we decrement by 1 a random set of entries of $\boldsymbol{P}$ and then use the branch-and-bound solver to reassign the corresponding rows optimally. We repeat this process until the average improvement between iterations drops below some threshold.

VI. Luby Transform Codes

In this section, we consider LT codes [16] for use in distributed computing. Specifically, we consider a distributed computing system where $\boldsymbol{\Psi}$ is an LT code encoding matrix, denoted by $\boldsymbol{\Psi}_{\mathsf{LT}}$, of fixed rate $\frac{m}{r}$. As explained in Section II, we divide the r coded rows of $\boldsymbol{C} = \boldsymbol{\Psi}_{\mathsf{LT}}\boldsymbol{A}$ into $\binom{K}{\eta q}$ disjoint batches, each of which is stored at a unique subset of size ηq of the K servers. For this scheme, due to the random nature of LT codes, we can assign coded rows to batches randomly. The distributed computation is carried out as explained in Section II-B, i.e., we wait for the fastest $g \geq q$ servers to complete their respective computations in the map phase, perform coded multicasting during the shuffle phase, and carry out the decoding of the N output vectors in the reduce phase.

Let Ω denote the degree distribution and $\Omega(d)$ the probability of degree d. Also, let $\bar{\Omega}$ be the average degree. Then, each row of the encoding matrix $\boldsymbol{\Psi}_{\mathsf{LT}}$ is constructed in the following manner. Uniformly at random select d unique entries of the row, where d is drawn from the distribution Ω. For each of these d entries, assign to it a nonzero element selected uniformly at random from $\mathbb{F}_{2^l}$. Specifically, we consider the case where Ω is the robust Soliton distribution parameterized by M and δ, where M is the location of the spike of the robust component and δ is a parameter for tuning the decoding failure probability for a given M [16].

A. Inactivation Decoding

We assume that decoding is performed using inactivation decoding [17]. Inactivation decoding is an efficient maximum likelihood decoding algorithm that combines iterative decoding with optimal decoding in a two-step fashion and is widely

used in practice. As suggested in [17], we assume that the optimal decoding phase is performed by Gaussian elimination. In particular, iterative decoding is used until the ripple is empty, i.e., until there are no coded symbols of degree 1, at which point an input symbol is inactivated. The iterative decoder is then restarted to produce a solution in terms of the inactivated symbol. This procedure is repeated until all input symbols are either decoded or inactivated. Note that the value of some input symbols may be expressed in terms of the values of the inactivated symbols at this point. Finally, optimal decoding of the inactivated symbols is performed via Gaussian elimination, and the decoded values are back-substituted into the decoded input symbols that depend on them. The decoding schedule has a large performance impact. Our implementation follows the recommendations in [17]. It is important to tune the parameters M and δ to minimize the number of inactivations.

Due to the nature of LT codes, we need to collect $m(1+\epsilon)$ intermediate values for each vector $\boldsymbol{y}$ before decoding. We refer to ϵ as the overhead. Under inactivation decoding, and for a given overhead ϵ, the probability of decoding failure with an overhead of at most ϵ, denoted by $P_{\mathsf{f}}(\epsilon)$, is lower bounded by [26]

$$P_{\mathsf{f}}(\epsilon) \geq \sum_{i=1}^{m} (-1)^{i+1} \binom{m}{i} \left(\sum_{d=1}^{m} \Omega(d) \frac{\binom{m-i}{d}}{\binom{m}{d}} \right)^{m(1+\epsilon)} . \quad (15)$$

Note that $P_{\mathsf{f}}(\epsilon)$ is the CDF for the random variable "decoding is not possible at a given overhead ϵ." Furthermore, the lower bound (15) well approximates the failure probability for an overhead slightly larger than $\epsilon = 0$. Denote by $F_{\mathsf{DS}}(\epsilon)$ the probability of decoding being possible at an overhead of at most ϵ. It follows that

$$F_{\mathsf{DS}}(\epsilon) = 1 - P_{\mathsf{f}}(\epsilon).$$

We find the decoding success probability density function (PDF) by numerically differentiating $F_{\mathsf{DS}}(\epsilon)$.

B. Code Design

We design LT codes for a minimum overhead ϵ_{min}, i.e., we collect at least $m(1+\epsilon_{\mathsf{min}})$ coded symbols from the servers before attempting to decode, and a target failure probability $P_{\mathsf{f,target}} = P_{\mathsf{f}}(\epsilon_{\mathsf{min}})$. We remark that increasing ϵ_{min} and $P_{\mathsf{f,target}}$ leads to a lower average degree $\bar{\Omega}$, and thus to less complex encoding and decoding and subsequently to a lower computational delay for encoding and decoding. The tradeoff is that the communication load increases as more intermediate values need to be transferred over the network on average. Furthermore, increasing ϵ_{min} and $P_{\mathsf{f,target}}$ may increase the average number of servers g required to decode. We thus need to balance the computational delay of the encoding and reduce phases against that of the map phase to achieve a low overall computational delay. Furthermore, waiting for more than $g = q$ servers typically increases the overall computational delay by more than what is saved by the less complex encoding and decoding given by the larger ϵ_{min} and $P_{\mathsf{f,target}}$. We thus choose ϵ_{min} and $P_{\mathsf{f,target}}$ such that decoding is possible with high probability using the number of coded rows stored at any set of q servers. Note that the overhead ϵ required for decoding may be larger than ϵ_{min}. We take this into account by numerically integrating the decoding success PDF multiplied by the performance of the scheme as a function of the overhead ϵ.

For a given ϵ_{min} and $P_{\mathsf{f,target}}$, we find a pair (M, δ) that minimizes the decoding complexity (see Section VI-C) under the constraint that $P_{\mathsf{f}}(\epsilon_{\mathsf{min}}) \approx P_{\mathsf{f,target}}$. Essentially, we minimize the computational delay of the reduce phase for a fixed delay of the map phase. We remark that LT codes with low decoding complexity have a low average degree $\bar{\Omega}$, and thus also low encoding complexity. Note that for a given M, decreasing δ lowers the failure probability, but also increases the decoding complexity. We find good pairs (M, δ) by selecting through binary search the largest M such that there exists a δ for which the lower bound on $P_{\mathsf{f}}(\epsilon_{\mathsf{min}})$ in (15) is approximately equal to $P_{\mathsf{f,target}}$. This heuristic produces codes with complexity very close to those found using basin-hopping [27] combined with the Powell optimization method [28].

C. Computational Delay

There are on average $\bar{\Omega}$ nonzero entries in each row of the LT code encoding matrix. The LT code encoding complexity is thus given by

$$\sigma_{\mathsf{encode,LT}} = \bar{\Omega} r n \sigma_{\mathsf{M}} + (\bar{\Omega} - 1) r n \sigma_{\mathsf{A}}.$$

We simulate the complexity of the decoding $\sigma_{\mathsf{reduce,LT}}$. Furthermore, we assume that the decoding complexity depends only on ϵ_{min}, i.e., we evaluate the decoding complexity only at $\epsilon = \epsilon_{\mathsf{min}}$, and simulate the number of servers g required for a given overhead ϵ.

D. Communication Load

The coded multicasting scheme (see Section II-B3) is designed for the case where we need m intermediate values per vector $\boldsymbol{y}$. Here, we tune it for the case where we instead need at least $m(1+\epsilon_{\mathsf{min}})$ intermediate values by increasing the number of coded multicast messages sent. Note that the coded multicasting scheme is greedy in the sense that it starts by multicasting coded messages to the largest possible number of recipients and then gradually lowers the number of recipients. Specifically, we perform the shuffle phase with (see (1))

$$s_{q,\mathsf{LT}} \triangleq \inf \left(s : \sum_{j=s}^{\eta q} \alpha_j \leq (1+\epsilon_{\mathsf{min}}) - \eta \right).$$

The communication load of the LT code-based scheme for a given $\epsilon \geq \epsilon_{\mathsf{min}}$ is then given by

$$L_{\mathsf{LT}} = \min \left(\sum_{j=s_{q,\mathsf{LT}}}^{\eta q} \frac{\alpha_j}{\phi(j)} + (1+\epsilon) - \eta - \sum_{j=s_{q,\mathsf{LT}}}^{\eta q} \alpha_j \, , \right.$$
$$\left. \sum_{j=s_{q,\mathsf{LT}}-1}^{\eta q} \frac{\alpha_j}{\phi(j)} + \max \left((1+\epsilon) - \eta - \sum_{j=s_{q,\mathsf{LT}}-1}^{\eta q} \alpha_j, 0 \right) \right).$$

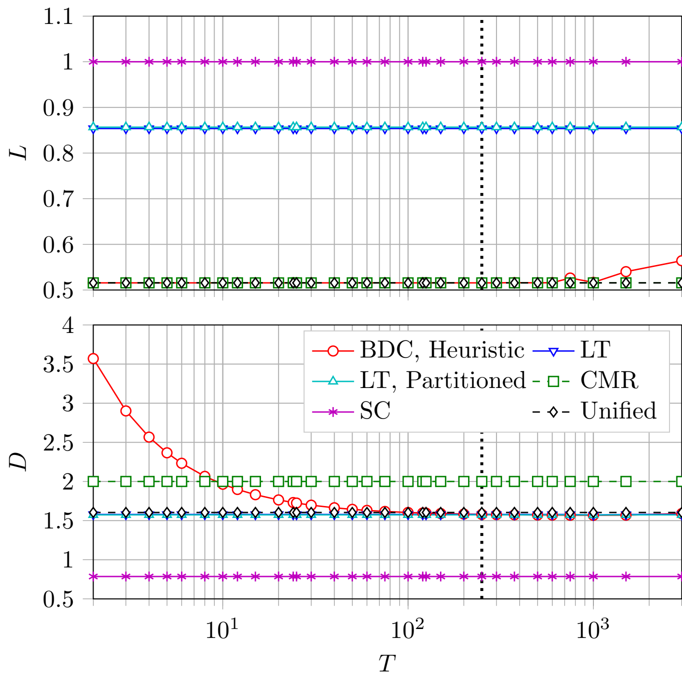

Fig. 5. The tradeoff between partitioning and performance for $m = 6000$, $n = 6000$, $K = 9$, $q = 6$, $N = 6000$, and $\eta = 1/3$.

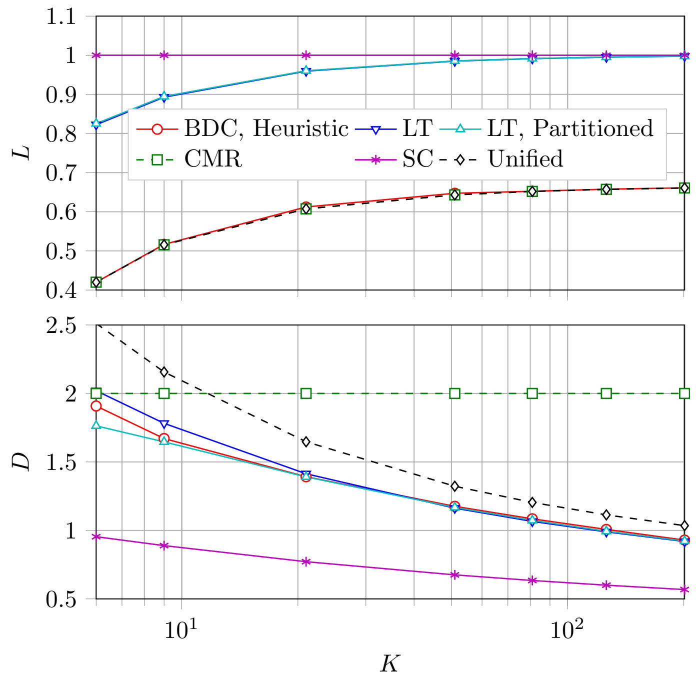

Fig. 6. Performance dependence on system size for $\eta q = 2$, $n = m/100$, $\eta m = 2000$, code rate $m/r = 2/3$, and $N = 500q$ vectors.

E. Partitioning of the LT Code-Based Scheme

We can apply partitioning to the LT code-based scheme in the same manner as for the BDC scheme. Specifically, we consider a block-diagonal encoding matrix $\mathbf{\Psi}_{\mathsf{BDC-LT}}$, where the blocks $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_T$ are LT code encoding matrices. In particular, we consider the case where the number of partitions T is equal to the partitioning limit of Theorem 1, i.e., $T = r/\binom{K}{\eta q}$. In this case the all-ones assignment matrix $\boldsymbol{P}$ introduced in the proof of Theorem 1 is a valid matrix. By using this assignment matrix and identical encoding matrices for each of the partitions, i.e., $\boldsymbol{\psi}_i = \boldsymbol{\psi}$, $i = 1, \ldots, T$, the encoding and decoding complexity of each partition is identical regardless of which set of servers $\mathcal{G}$ first completes the map phase. Furthermore, by the same argument as in the proof of Theorem 1, we are guaranteed that if any partition can be decoded using the coded rows stored at the set of servers $\mathcal{G}$, all other partitions can also be decoded.

VII. Numerical Results

We present numerical results for the proposed BDC and LT code-based schemes and compare them with the schemes in [7]–[9]. Furthermore, we compare the performance of the BDC scheme with assignment $\boldsymbol{P}$ produced by the heuristic and hybrid solvers. We also evaluate the performance of the LT code-based scheme for different $P_{\mathsf{f,target}}$ and ϵ_{min}. For each plot, the field size is equal to one more than the largest number of coded rows considered for that plot, $r+1$, rounded up to the closest power of 2. The results, except those in Fig. 12, are normalized by the performance of the uncoded scheme. Unless stated otherwise, the assignment $\boldsymbol{P}$ is given by the heuristic solver. As in [9], we assume that $\phi(j) = j$.

A. Coded Computing Comparison

In Fig. 5, we depict the communication load L (see Definition 1) and the computational delay D (see Definition 2) as a function of the number of partitions, T. The system parameters are $m = 6000$, $n = 6000$, $K = 9$, $q = 6$, $N = 6000$, and $\eta = 1/3$. The parameters of the CMR and SC schemes are $q_{\mathsf{CMR}} = 9$, $\eta_{\mathsf{CMR}} = \frac{2}{9}$, and $\eta_{\mathsf{SC}} = \frac{1}{6}$. The minimum overhead for the LT code-based scheme is $\epsilon_{\mathsf{min}} = 0.3$ and its target failure probability is $P_{\mathsf{f,target}} = 0.1$. For up to $r/\binom{K}{\eta q} = 250$ partitions (marked by the vertical dotted line), the BDC scheme does not incur any loss in D_{map} and communication load with respect to the unified scheme (see Theorem 1). Furthermore, the BDC scheme yields about a 2% lower delay compared to the unified scheme for $T = 1000$. The delay of the LT code-based scheme is slightly worse than that of the BDC scheme, and the load is about 65% higher (for $T = 250$). Partitioning the LT code-based scheme increases the communication load and reduces the computational delay by about 0.5%. We remark that the number of partitions for the LT code-based scheme is fixed at $r/\binom{K}{\eta q}$. For heavy partitioning of the BDC scheme, a tradeoff between partitioning level, communication load, and map phase delay is observed. For example, with 3000 partitions (the maximum possible), there is about a 10% increase in communication load over the unified scheme. Note that the gain in computational delay saturates, thus there is no reason to partition beyond a given level. The load of the SC scheme is about twice that of our proposed schemes and the delay is about half. Finally, the delay of the BDC and the LT code-based scheme is about 25% lower compared to the CMR scheme for $T > 100$.

In Fig. 6, we plot the performance for a constant $\eta q = 2$, $n = m/100$, $\eta m = 2000$, code rate $m/r = 2/3$, and $N = 500q$ vectors as a function of the number of servers, K. The ratio m/n is motivated by machine learning applications,

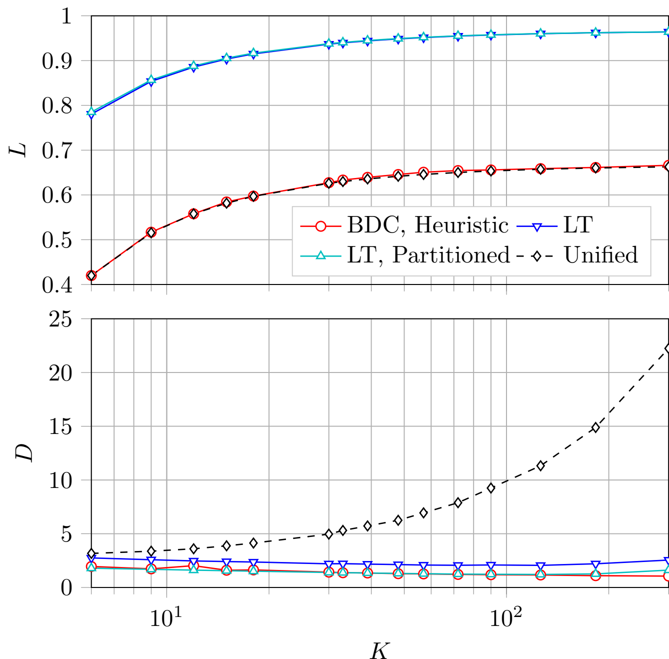

Fig. 7. Performance dependence on system size with constant complexity of the map phase per server, $m/r = 2/3$, $\eta q = 2$, $n = m/100$, and $N = n$.

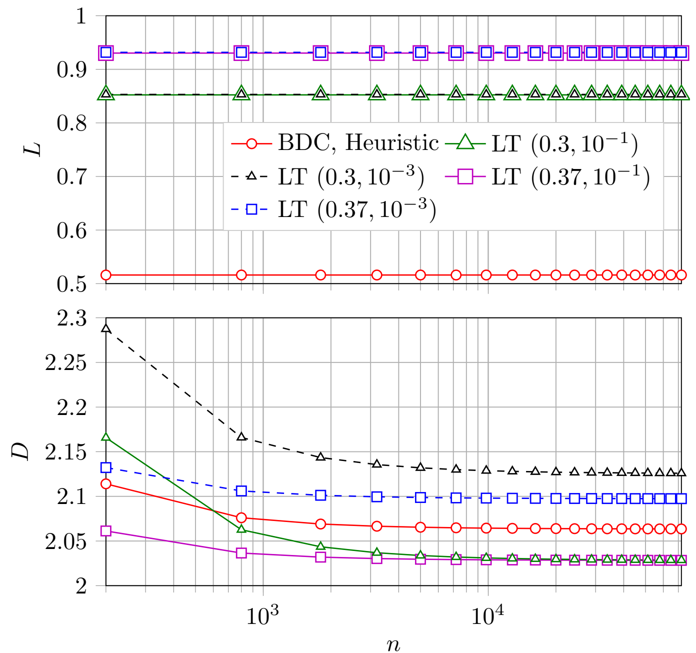

Fig. 8. Performance dependence on the number of columns n of $\boldsymbol{A}$ for $m = 2400$, $K = 9$, $q = 6$, $N = 60$, $T = 240$, and $\eta = 1/3$. The parameters of the LT code-based scheme are given in the legend as $(\epsilon_{\text{min}}, P_{\text{f,target}})$.

where the number of rows and columns often represent the number of samples and features, respectively. Note that the number of arithmetic operations performed by each server in the map phase increases with K. We choose the number of partitions T that minimizes the delay under the constraint that the communication load is at most 1% higher compared to the unified scheme. The parameters of the LT code-based scheme are $\epsilon_{\text{min}} = 0.335$ and $P_{\text{f,target}} = 0.1$. The results shown are averages over 1000 randomly generated realizations of $\mathcal{G}$. Our proposed BDC scheme outperforms the unified scheme in terms of computational delay by between about 25% (for $K = 6$) and 10% (for $K = 201$). Furthermore, the delay of both the BDC and LT code-based schemes are about 50% lower than that of the CMR scheme for $K = 201$. For $K = 6$ the computational delay of the unpartitioned and partitioned LT code-based schemes is about 5% higher and 8% lower compared to the BDC scheme, respectively. For $K = 201$ the delay of the LT code-based scheme is about 1% lower than that of the BDC scheme. However, the communication load is about 45% higher. Finally, the communication load of the BDC scheme is between about 42% (for $K = 6$) and 66% (for $K = 201$) of that of the SC scheme.

In Fig. 7, we show the performance for code rate $m/r = 2/3$, $\eta q = 2$, and a fixed workload per server as a function of K. Specifically, we fix the number of additions and multiplications computed by each server in the map phase to 10^8 (±5% to find valid parameters) and scale m, n, N with K. The number of rows m of $\boldsymbol{A}$ takes values between 12600 and 59800, and we let $n = m/100$ and $N = n$. The number of partitions T is selected in the same way as for Fig. 6. The results shown are averages over 1000 randomly generated realizations of $\mathcal{G}$. The computational delay of the unified scheme is about a factor 20 higher than that of the BDC scheme for $K = 300$. The computational delay of the partitioned LT code-based scheme is similar to that of the BDC scheme, while the delay of the unpartitioned LT code-based scheme is about 60% higher. Furthermore, the communication load of the LT code-based scheme is about 45% higher compared to those of the unified and BDC schemes.

In Fig. 8, we plot the performance of the BDC and LT code-based schemes as a function of the number of columns n. The system parameters are $m = 2400$, $K = 9$, $q = 6$, $N = 60$, $T = 240$, and $\eta = 1/3$. The communication load of the LT code-based scheme depends primarily on the minimum overhead ϵ_{min} and the computational delay primarily on the target failure probability $P_{\text{f,target}}$. We remark that a higher $P_{\text{f,target}}$ allows for using codes with lower average degree and thus less complex encoding and decoding. For $n = 20000$, the computational delay of the LT code-based scheme with $P_{\text{f,target}} = 0.1$ is about 1.5% lower than that of the BDC scheme. For $P_{\text{f,target}} = 0.001$, the computational delay is about 3% and 1.5% higher than that of the BDC scheme when $\epsilon_{\text{min}} = 0.3$ and $\epsilon_{\text{min}} = 0.37$, respectively. On the other hand, the communication load of the LT code-based scheme with $\epsilon_{\text{min}} = 0.3$ and $\epsilon_{\text{min}} = 0.37$ is about 41% and 44% higher than that of the BDC scheme, respectively.

B. Assignment Solver Comparison

In Figs. 9 and 10, we plot the performance of the BDC scheme with assignment $\boldsymbol{P}$ given by the heuristic and the hybrid solver. We also give the average performance over 100 random assignments. The vertical dotted line marks the partitioning limit of Theorem 1. The parameters in Figs. 9 and 10 are identical to those in Figs. 5 and 6, respectively.

In Fig. 9, we plot the performance as a function of the number of partitions, T. For T less than about 200, the performance for all solvers is identical. On the other hand, for

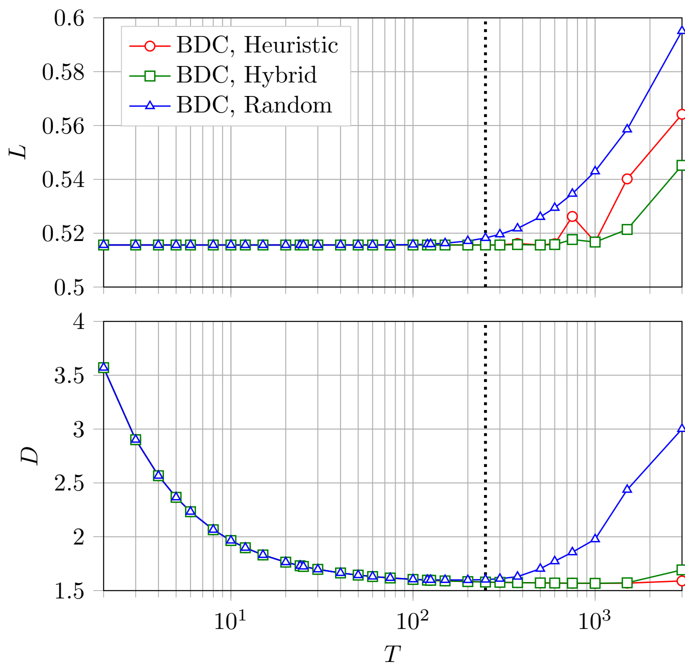

Fig. 9. Solver performance as a function of partitioning for $m = 6000$, $n = 6000$, $K = 9$, $q = 6$, $N = 6000$, and $\eta = 1/3$.

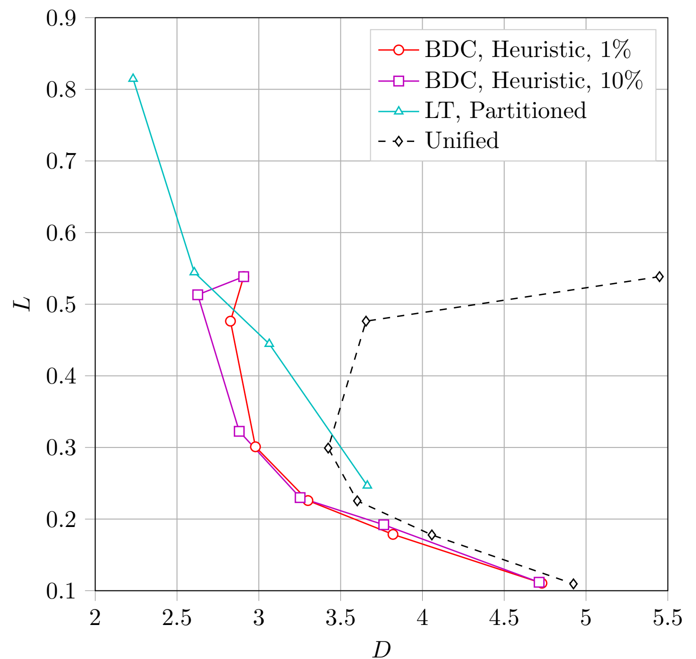

Fig. 11. The tradeoff between communication load and computational delay for $K = 14$, $m = 50000 \pm 3\%$, $n = 500$, $N = 840$, and $\eta = 1/2$.

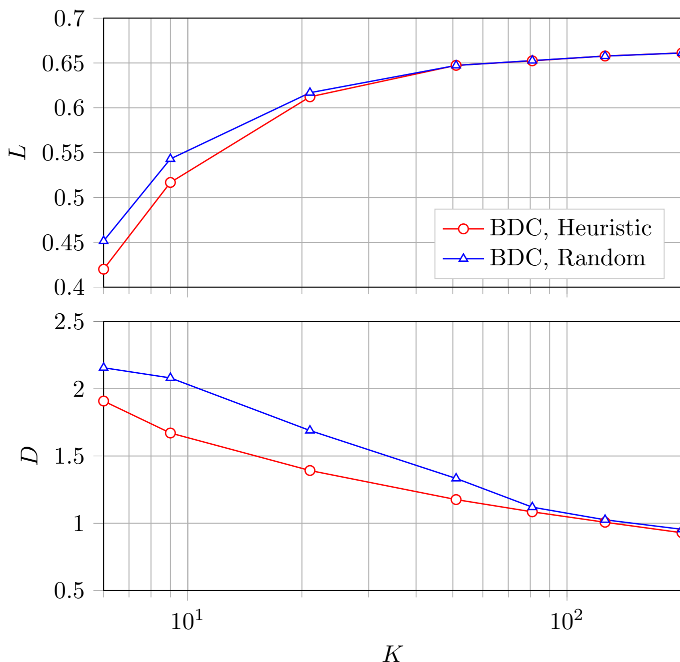

Fig. 10. Solver performance as a function of system size for $\eta q = 2$, $n = m/100$, $\eta m = 2000$, code rate $m/r = 2/3$, and $N = 500q$ vectors.

$T > 200$ both the computational delay and the communication load are reduced with $\boldsymbol{P}$ from the heuristic solver over the random assignments (about 5% for load and 47% for delay at $T = 3000$). A further improvement in communication load can be achieved using the hybrid solver, but at the expense of a possibly larger computational delay.

In Fig. 10, we plot the performance as a function of the number of servers, K. The results shown are averages over 1000 randomly generated realizations of $\mathcal{G}$. For $K = 6$, the communication load of the heuristic solver is about 5% lower than that of the random assignments, but for $K = 201$ the difference is negligible. In terms of computational delay, the heuristic solver outperforms the random assignments by about 18% and 3% for $K = 9$ and $K = 201$, respectively. The hybrid solver is too computationally complex for use with the largest systems considered.

C. Tradeoff Between Communication Load and Computational Delay

In Fig. 11, we show the tradeoff between communication load and computational delay. The parameters are $K = 14$, $m = 50000$ ($\pm 3\%$ to find valid parameters), $n = 500$, $N = 840$, and $\eta = 1/2$. Note that the code rate is decreasing toward the bottom of the plot. We select the number of partitions T that minimizes the delay while the load is at most 1% or 10% higher compared to the unified scheme. Allowing a 10% increased load gives up to about 7% lower delay compared to allowing a 1% increase. For the topmost data point of the BDC and unified schemes the encoding complexity dominates, and there is no reason to operate at this point since both the delay and load can be reduced. The parameters of the partitioned LT code-based scheme are $\epsilon_{\mathsf{min}} = 0.3$ and $P_{\mathsf{f,target}} = 10^{-1}$. For the data point with minimum computational delay, the LT code-based scheme yields about 15% lower delay at the expense of about a 30% higher load compared to the BDC scheme. Finally, the computational delay of the BDC scheme is between about 47% and 4% lower compared to the unified scheme for the topmost and bottommost data points, respectively.

D. Computational Delay Deadlines

In Fig. 12, we plot the probability of a computation not finishing before a deadline t, i.e., the probability of the

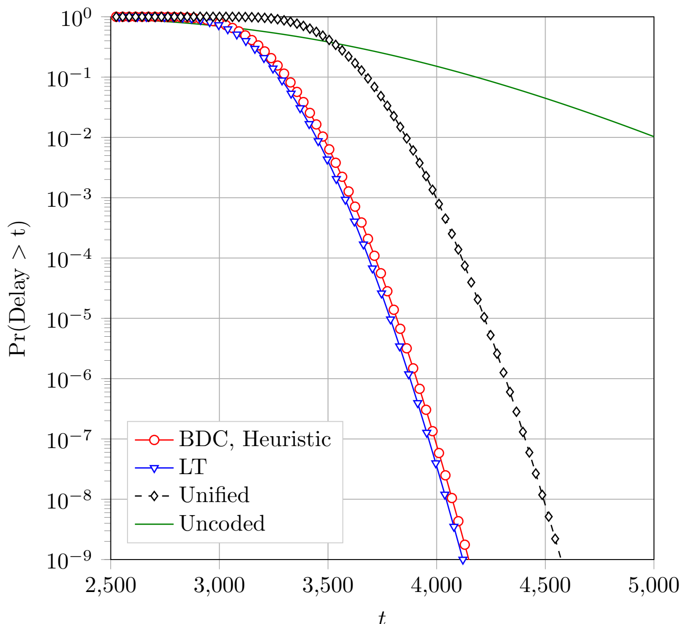

Fig. 12. The probability of a computation not finishing before a deadline t for $K = 201$, $q = 134$, $m = 134000$, $n = 1340$, $N = 67000$ vectors, $T = 6700$ partitions, code rate $m/r = 2/3$, $\epsilon_{\text{min}} = 0.335$, and $P_{\text{f,target}} = 10^{-9}$.

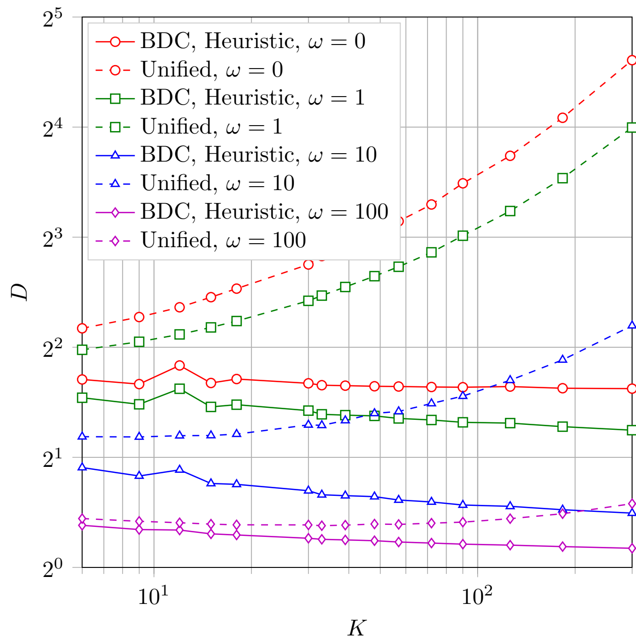

Fig. 13. Computational delay as a function of system size for varying scale of the tail of the runtime distribution. The system parameters and communication load are identical to those in Fig. 7.

computational delay being larger than t. As in [29], we plot the complement of the CDF of the delay in logarithmic scale. On the horizontal axis, we show the deadline t. The system parameters are $K = 201$, $q = 134$, $m = 134000$, $n = 1340$, $N = 67000$ vectors, $T = 6700$ partitions, and code rate $m/r = 2/3$. The parameters for the LT code-based scheme are $\epsilon_{\text{min}} = 0.335$ and $P_{\text{f,target}} = 10^{-9}$. The results are due to simulations. In particular, we simulate the decoding failure probability of LT codes for various t and extrapolate from these points under the assumption that the decoding failure probability is Gamma distributed. The fitted values deviate negligibly from the simulated values.

When the deadline is $t = 3500$, the probability of exceeding the deadline is about 0.4 for the unified and uncoded schemes. For the BDC scheme the probability is only about $7 \cdot 10^{-3}$. The probability is slightly lower for the LT code-based scheme, about $4 \cdot 10^{-3}$. If we instead consider a deadline $t = 4000$, the probability of exceeding the deadline is about 10^{-3} and 0.15 for the unified and uncoded schemes, respectively. For the BDC scheme the probability of exceeding the deadline is about $9 \cdot 10^{-8}$, i.e., 4 orders of magnitude lower compared to the unified scheme. The LT code-based scheme further improves the performance with a probability of exceeding the deadline of about $3 \cdot 10^{-8}$. We remark that for the data point with minimum delay in Fig. 11, the LT code-based scheme has a significant advantage over the BDC scheme in terms of meeting a short deadline.

E. Alternative Runtime Distribution

Here, we consider a runtime distribution with CDF

$$F_H(h; \sigma) = \begin{cases} 1 - \mathrm{e}^{-(h-\sigma)/\beta}, & \text{for } h \geq \sigma \\ 0, & \text{otherwise} \end{cases},$$

where σ is the shift and β is a parameter that scales the tail of the distribution, i.e., it differs from the one considered previously by that the scale of the tail may be different from the shift. It is equal to the previously considered distribution if $\beta = \sigma$. This model has been used to model distributed computing in, e.g., [30]. Under this model we assume that the reduce delay of the uncoded scheme follows the distribution above with parameters β and $\sigma_{\text{UC,reduce}} = 0$ since each server has to assemble the final output from the intermediate results regardless coding is used or not. We assume that the encoding delay of the uncoded scheme is zero. Denote by σ_{c} the computational complexity of matrix-vector multiplication for the BDC and unified schemes. We let $\beta = \omega\sigma_{\text{c}}$ for $\omega = 0, 1, 10, 100$. In Fig. 13, we plot the computational delay normalized by that of the uncoded scheme. The system parameters (and thus also the communication load) are identical to those in Fig. 7.

We observe the greatest gain of the BDC scheme over the unified scheme for small ω since the benefits of straggler coding are small compared to the added delay due to encoding and decoding, which is significant for the unified scheme. For larger ω the benefits of straggler coding are larger while the delay due to encoding and decoding remains constant. Hence, the performance of both schemes converge. However, even for $\omega = 100$ the delay of the unified scheme is about 33% higher than that of the BDC scheme for the largest system considered ($K = 300$). We remark that for the example considered in [30] the parameters $\beta = \sigma = 1$, i.e., $\omega = 1$, are used.

VIII. Conclusion

We introduced two coding schemes for distributed matrix multiplication. One is based on partitioning the matrix into submatrices and encoding each submatrix separately using MDS codes. The other is based on LT codes. Compared to

the earlier scheme in [9] and to the CMR scheme in [7], both proposed schemes yield a significantly lower overall computational delay. For instance, for a matrix of size 59800×598, the BDC scheme reduces the computational delay by about a factor 20 over the scheme in [9] with about a 1% increase in communication load. The LT code-based scheme may reduce the computational delay further at the expense of a higher communication load. For example, for a matrix with about 50000 rows, the computational delay of the LT code-based scheme is about 15% lower than that of the BDC scheme with a communication load that is about 30% higher. Finally, we have shown that the proposed coding schemes significantly increase the probability of a computation finishing within a deadline. The LT code-based scheme may be the best choice in situations where high reliability is needed due to its ability to decrease the computational delay at the expense of the communication load.

Acknowledgment

The authors would like to thank Dr. Francisco Lázaro and Dr. Gianluigi Liva for fruitful discussions and insightful comments on LT codes.

References

[1] L. A. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2009.

[2] C. L. P. Chen and C.-Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on big data,” *Information Sciences*, vol. 275, pp. 314–347, Aug. 2014.

[3] L. A. Barroso, “Warehouse-scale computing: The machinery that runs the cloud,” in *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2010 Symposium*. Washington, DC: The National Academies Press, 2011, pp. 15–19.

[4] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in *Proc. Conf. Symp. Operating Systems Design & Implementation*, San Francisco, CA, Dec. 2004, p. 10.

[5] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, “Apache Spark: A unified engine for big data processing,” *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, Nov. 2016.

[6] R. Ranjan, “Streaming big data processing in datacenter clouds,” *IEEE Cloud Computing*, vol. 1, no. 1, pp. 78–83, May 2014.

[7] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “Coded MapReduce,” in *Proc. Allerton Conf. Commun., Control, and Computing*, Monticello, IL, Sep./Oct. 2015, pp. 964–971.

[8] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[9] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “A unified coding framework for distributed computing with straggling servers,” in *Proc. Work. Network Coding and Appl.*, Washington, DC, Dec. 2016.

[10] H. Ishii and R. Tempo, “The PageRank problem, multiagent consensus, and web aggregation: A systems and control viewpoint,” *IEEE Control Systems Mag.*, vol. 34, no. 3, pp. 34–53, Jun. 2014.

[11] K. Lee, C. Suh, and K. Ramchandran, “High-dimensional coded matrix multiplication,” in *Proc. IEEE Int. Symp. Inf. Theory*, Aachen, Germany, Jun. 2017, pp. 2418–2422.

[12] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, “Polynomial codes: an optimal design for high-dimensional coded matrix multiplication,” in *Proc. Advances Neural Inf. Processing Systems*, Long Beach, CA, Dec. 2017, pp. 4403–4413.

[13] S. Dutta, V. Cadambe, and P. Grover, “Short-Dot: Computing large linear transforms distributedly using coded short dot products,” in *Proc. Advances Neural Inf. Processing Systems*, Barcelona, Spain, Dec. 2016, pp. 2100–2108.

[14] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, “Coded computation over heterogeneous clusters,” in *Proc. IEEE Int. Symp. Inf. Theory*, Aachen, Germany, Jun. 2017, pp. 2408–2412.

[15] A. Severinson, A. Graell i Amat, and E. Rosnes, “Block-diagonal coding for distributed computing with straggling servers,” in *Proc. IEEE Inf. Theory Work.*, Kaohsiung, Taiwan, Nov. 2017, pp. 464–468.

[16] M. Luby, “LT codes,” in *Proc. IEEE Symp. Foundations Computer Science*, Vancouver, BC, Canada, Nov. 2002, pp. 271–280.

[17] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, “RaptorQ Forward Error Correction Scheme for Object Delivery,” Internet Requests for Comments, RFC Editor, RFC 6330, Aug. 2011.

[18] J. Edmonds and M. Luby, “Erasure codes with a hierarchical bundle structure,” *IEEE Trans. Inf. Theory*, 2017, to appear.

[19] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at Google with Borg,” in *Proc. European Conf. Computer Systems*, Bordeaux, France, Apr. 2015.

[20] G. Liang and U. C. Kozat, “TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes,” in *Proc. IEEE Conf. Computer Commun.*, Toronto, ON, Canada, Apr./May 2014, pp. 826–834.

[21] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja, *A First Course in Order Statistics*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008.

[22] C. Walck, “Hand-book on statistical distributions for experimentalists,” Particle Physics Group, University of Stockholm, Sweden, Tech. Rep. SUF-PFY/96-01, Sep. 2007. [Online]. Available: http://staff.fysik.su.se/~walck/suf9601.pdf

[23] S.-J. Lin, T. Y. Al-Naffouri, Y. S. Han, and W.-H. Chung, “Novel polynomial basis with fast Fourier transform and its application to Reed-Solomon erasure codes,” *IEEE Trans. Inf. Theory*, vol. 62, no. 11, pp. 6284–6299, Nov. 2016.

[24] G. Garrammone, “On decoding complexity of Reed-Solomon codes on the packet erasure channel,” *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 773–776, Apr. 2013.

[25] A. Severinson, “Coded Computing Tools,” Aug. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1400313

[26] B. Schotsch, G. Garrammone, and P. Vary, “Analysis of LT codes over finite fields under optimal erasure decoding,” *IEEE Commun. Lett.*, vol. 17, no. 9, pp. 1826–1829, Sep. 2013.

[27] D. J. Wales and J. P. K. Doye, “Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms,” *J. Phys. Chem. A*, vol. 101, no. 28, pp. 5111–5116, Jul. 1997.

[28] M. J. D. Powell, “An efficient method for finding the minimum of a function of several variables without calculating derivatives,” *The Computer Journal*, vol. 7, no. 2, pp. 155–162, Jan. 1964.

[29] S. Dutta, V. Cadambe, and P. Grover, “Coded convolution for parallel and distributed computing within a deadline,” in *Proc. IEEE Int. Symp. Inf. Theory*, Aachen, Germany, Jun. 2017, pp. 2403–2407.

[30] D. Wang, G. Joshi, and G. Wornell, “Using straggler replication to reduce latency in large-scale parallel computing,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 3, pp. 7–11, Dec. 2015.